PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-187463
(43)Date of publication of application : 21.07.1998

(51)Int.Cl. G06F 9/45
G06F 9/44

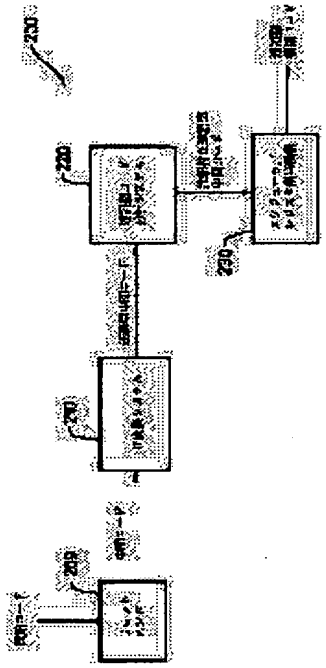
(21)Application number : 09-320823
(22)Date of filing : 21.11.1997

(71)Applicant : HEWLETT PACKARD CO <HP>
(72)Inventor : JOHNSON RICHARD C
SCHLANSKER MICHAEL S

(30)Priority
Priority number : 96 756423 Priority date : 26.11.1996 Priority country : US

(54) COMPILER

(57)Abstract:
PROBLEM TO BE SOLVED: To optimize the performance at predicate code execution time by analyzing the relation between predicates of a predicate type code by handling and matching the predicate expression of the predicate type code and analyzing data flow characteristics of the predicate type code.
SOLUTION: The compiler 200 includes an if conversion system 210 connected to a front end 209, and also includes a predicate type code analysis system 220 connected to a system 210 and a schedule and register allocating mechanism 230 which is connected to the system 220. Then the system 220 is connected directly to the system 210, and receives a predicate type intermediate code and directly analyzes it. Consequently, scheduling restrictions on the predicate type code are added as a comment to the code prior to scheduling and register allocation and the analysis can be made not to depend upon the control flow of an original source code.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平10－187463

(43)公開日　平成10年(1998)７月21日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/45		Ｇ０６Ｆ	9/44	３２２Ｆ
	9/44	５２０			５２０Ｐ

審査請求　未請求　請求項の数１　ＯＬ　（全 25 頁）

(21)出願番号　特願平9－320823

(22)出願日　平成９年(1997)11月21日

(31)優先権主張番号　７５６，４２３

(32)優先日　1996年11月26日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　590000400
ヒューレット・パッカード・カンパニー
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000

(72)発明者　リチャード・シー・ジョンソン
アメリカ合衆国95014カリフォルニア州クパーチノ、エヌ・フットヒル・プルバード　10230、ナンバー・イー　14

(72)発明者　マイケル・エス・シュランスカー
アメリカ合衆国94024カリフォルニア州ロス・アルトス、ラ・プレンダ　409

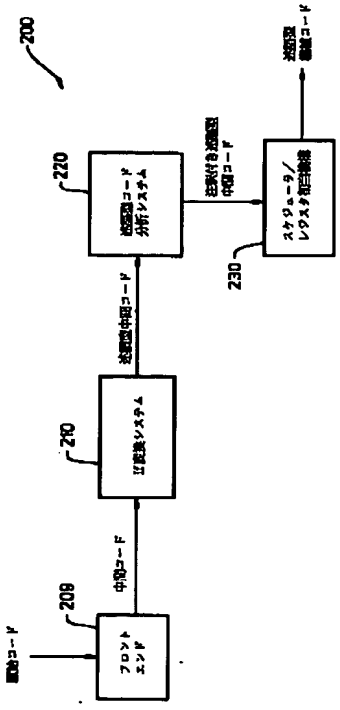
(74)代理人　弁理士　岡田　次生

(54)【発明の名称】　コンパイラ

(57)【要約】

【課題】述語型コードにおける述語間の関係を分析して述語コードの実行時性能を最適化するコンパイラを提供する。

【解決手段】述語型コードの述語表現を取り扱いそして照会を行うことによって述語型コードのデータ・フロー特性を分析するデータ・フロー分析システムを含む述語型コンパイラを提供する。該コンパイラは、述語型コードの局所的／大域的述語関係を記憶しその局所的／大域的述語関係に関する照会に回答する述語照会システムを含む述語反応分析機構を備え、それによって、コンパイルされるコードの最適なスケジューリングおよびレジスタ割り当てが達成される。



1

【特許請求の範囲】

【請求項１】述語型コードをコンパイルするコンパイラであって、

述語型コードの述語表現を取り扱い照会を行うことにより述語型コードのデータ・フロー特性を分析するデータ・フロー分析システム、

を備えるコンパイラ。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、コンピュータ・システムにおけるプログラミング言語コンパイラに関するもので、特に、述語型コードの直接分析を含む述語型コード・コンパイラに関するものである。

【０００２】

【従来の技術】周知の通り、コンピュータ・システムは典型的には、中央処理装置(ＣＰＵ)またはマイクロプロセッサとして知られているプロセッサを１つまたは複数含む。プロセッサは、典型的には、コンピュータ・システムにおける種々のタスクを実行するためソフトウェア・プログラムの命令を実行する。プロセッサは機械語だけしか理解し解釈することができないので、ソフトウェア・プログラムの命令は機械語形式(すなわちバイナリ形式)をしている。機械語命令は、本明細書において、以下、機械コードまたは目的コードと呼ぶ。

【０００３】機械語は書いたり理解するのが非常に困難であるので、ソフトウェア・プログラムの命令を人間が読める形式でコード化または定義する(Ｃおよびフォートランのようなⅰ)高水準原始プログラミング言語が開発された。そような原始プログラミング言語ソフトウェア・プログラムを原始コードと呼ぶ。原始コードは、プロセッサによって実行される前にコンパイラ・プログラムによって機械コードに変換または翻訳される必要がある。

【０００４】初期の先行技術プロセッサは、典型的には、単一命令単一データ(すなわちsingle instruction single dataであって以下頭文字をとってＳＩＳＤと略称する)プロセッサである。典型的ＳＩＳＤプロセッサは、単一の命令ストリームおよび単一のデータ・ストリームを受け取る。ＳＩＳＤプロセッサは、各命令を順次実行し、単一の記憶域にあるデータを処理する。しかしながら、このようなＳＩＳＤプロセッサのアアーキテクチャは、高い処理スループットを達成する上で障害を持つ。

【０００５】プロセッサの処理スループットを増加させるため、多くの並列処理アーキテクチャが開発されてきた。そのような並列処理モデルの１つのタイプは、命令レベルの平行プロセッサとして知られている。このようなプロセッサは、instruction-level parallelの頭文字をとってＩＬＰプロセッサと呼ばれる。ＩＬＰプロセッサにおいては、(スケジューリングおよび同期決定を行

2

う)計算処理の基本単位は、個々のadd(加算)、multiply(乗算)、load(ロード)またはstore(記憶)のようなプロセッサ命令である。相互に依存関係のない命令が並列的にロードされ実行される。ＩＬＰプロセッサを使用すれば、プログラム実行の間、命令のスケジューリングまたは同期決定を行う必要はない。決定の一部は、プログラムのコンパイルの間に行うことができる。例えば、コンパイラが２つの動作が独立している(すなわちどちらも入力として他方の結果を必要としない)ということを証明することができれば、それら動作は並列に実行することができる。

【０００６】しかしながら、プログラム・コードの中には頻繁にかつ予測できない分岐命令があって、大量の命令レベルの並列化の開拓にとって主要な障壁となる場合がある。これは、分岐命令の中には、典型的には、分岐遅延または予測エラー負荷を伴うものがあり、そのため実行時に処理が停止することがあるからである。加えて、分岐命令は、典型的には、命令レベルの平行に関する限りコードのスケジューリング範囲を制限する。分岐動作は、分岐と呼ばれる。

【０００７】分岐を除去し、命令レベル並列化を向上させるため、新しいアーキテクチャ・モデルが提案されている。そのようなモデルにおいては、各プロセッサ演算は、ブール値を持つソース・オペレンドによってガードされる。オペランドの値は、演算が実行されるかまたは無効にされるかを決定する。このようなアーキテクチャ・モデルは述語型実行と呼ばれ、ブール値を持つソース・オペランドは、述語と呼ばれる。命令セット・アーキテクチャの観点から見れば、述語型実行の主要機構は、各演算をガードする述語および述語を計算処理するために使用される一組のcompare-to-predicate演算である。述語型実行は、典型的には、多くの分岐を完全に除去し、基本ブロックの中でコードを移動させる規則を一般化させている。多くの場合、全非周期的な制御フロー・サブグラフが単一で分岐のないコード・ブロックに変換される。

【０００８】分岐を適切な述語計算およびガードと置き換えるプロセスは、ｉｆ変換または述語変換と呼ばれる。ｉｆ変換の結果として生ずるコードは、述語型コードと呼ばれる。図１のＡは、図１のＢに表されるプログラムに関する伝統的コードを示す。図１のＣは、図１のＡの伝統的コードから変換された述語型コードを示す。図１のＣから観察されるように、実行を制御する明示的分岐は、演算に関するガード述語、および、適切な述語値を計算するcompare-to-predicate演算と置き換えられている。すべての非分岐命令は、ｉｆ変換の間に述語化されている。結果は、単一で分岐のない述語型コードとなっている。

【０００９】

【発明が解決しようとする課題】しかしながら、述語型

コードが従来型コンパイラによってコンパイルされる時問題が発生する。これは、従来型コンパイラのデータ・フロー分析ツールが、典型的には、述語型コードをコンパイルする時述語の間の関係を分析しないためである。図２は、述語型コードをコンパイルするための従来技術のコンパイラ５０を示す。図２からわかるように、コンパイラ５０は、ｉｆ変換システム５１、スケジューラ／レジスタ割当機構５２およびデータ・フロー分析システム５３を含む。しかしながら、データ・フロー分析システム５３は、オリジナル・コードのデータ依存性を分析するだけで、述語型コードの述語間の関係に関する情報をそのデータ・フロー分析へ組み入れない。このため、典型的には、データ・フロー分析システムは、述語型コードの実行時動作について不正確な仮定を行うか、あるいは、何ら仮定を行わずそのためスケジューリングやレジスタ割り当てのような重大な分野で過度に保守的結果を招くこととなる。図３は、図２のコンパイラによって生成される図１のＣのコードの保守的スケジューリングを示す。図３は、述語型コードをコンパイルするため従来技術のアプローチを使用する場合の欠点を示している。従って、述語型コードにおける述語間の関係を分析して述語コードの実行時性能を最適化するコンパイラが必要とされている。

【００１０】

【課題を解決するための手段】本発明の１つの特徴は、述語型コードのコンパイルを最適化することである。本発明の別の特徴は、述語型コードの実行時性能を向上させることである。本発明の更に別の特徴は、述語型コードをコンパイルする時述語型コードの述語関係を開拓するコンパイラを提供することである。更にまた、本発明の１つの特徴は、述語型コードの述語関係を記憶する述語型コード・コンパイラに関する述語照会システムを提供し、それによってコンパイラが述語関係を使用して、スケジューリング、割り当ておよび最適化などのデータ・フロー分析を実施することを可能にする点である。

【００１１】本発明の述語型コード・コンパイラは述語型コードの述語表現を取り扱いそして照会を行うことによって述語型コードのデータ・フロー特性を分析するデータ・フロー分析システムを含む。本発明は、更に、述語型コードをコンパイルするコンパイラのための述語反応分析機構を提供する。該述語反応分析機構は、述語型コードの局所的／大域的述語関係を記憶しその局所的／大域的述語関係に関する照会に回答する述語照会システムを含む。

【００１２】

【発明の実施の形態】本発明のいくつかの実施形態を説明する以下の詳細な記述は、記述を明解にするため特定の用語を使用する。しかしながら、本発明は使用されるそれら特定の用語に限定されるものではなく、むしろ、実質的に同等の成果を達成するように実質的に同等の形

態で動作する技術的に等価なものすべてを含む。

【００１３】図４は、述語型実行をサポートするプロセッサ１０２を含むコンピュータ・システム１００を示す。述語型実行の主な特徴は、付加的ブール・オペランドおよび述語を計算処理する一組のcompare-to-predicate(対述語比較)演算命令である。述語型実行をサポートするため、プロセッサ１０２は、述語を記憶する一組の述語レジスタ(図示されていない)を含む。述語レジスタ・ファイルは、命令形式拡張としてプロセッサ１０２のＡＬＵ(すなわち算術論理ユニット)(これもまた図示されていない)に接続している。述語レジスタ・ファイルの各レジスタは、演算または命令の実行をガードするために使用される述語入力(すなわち述語)を記憶する。そのような演算は述語型演算と呼ばれる。演算に関する述語が１(すなわち真)の値を持つ時、演算は通常通り実行する。述語がゼロ(すなわち偽)の値を持つ時、演算の実行は無効にされる。これは、機械状態の変化が起こらないことを意味する。

【００１４】加えて、プロセッサ１０２の命令セットは、compare-to-predicate演算を含む。compare-to-predicate演算は述語を計算処理して、その結果を述語レジスタ・ファイルの述語レジスタに書き込む。compare-to-predicate演算の各々は、２つの述語レジスタに書き込むことができる。

【００１５】本発明の１つの実施形態において、プロセッサ１０２によって実行されるcompare-to-predicate演算の各々は、以下の形式を持つ。

P1, P2=cmpp <d1> <d2> (r1 <cond> r2) if q

上記形式において、

・ p1およびp2は、宛先述語レジスタ・オペランド(すなわち述語)である。

・ cmppは、比較命令コードの総称である。

・ <d1>および<d2>は、比較演算に関するアクションおよびモードを指定する２字記述子である。例えば、アクションは、記述子の最初の文字によって指定される無条件(u)、条件つき(c)、並列ＯＲ(o)および並列ＡＮＤ(a)のアクションを含む。各アクションは、記述子の２番目の文字によって指定される通常モード(n)および補完モード(c)を持つ。

・ "r1 <cond> r2"は、実際の比較を指定する。例えば、<cond>は、"<"(より小)、">"(より大)、"="(等しい)、"≦"(以下)または"≧"(以上)である。

・ qは、比較演算に関する述語レジスタ・オペランドである。qはガード述語として出現するが、それは比較演算データ入力としてのみ認識されるだけである。プロセッサ１０２はcompare-to-predicate演算に関して上記以外の実行形式をサポートすることもできるであろう。

【００１６】述語型実行は、プロセッサ１０２上で動く従来型コードが述語型コード(例えば図１のＣの述語型コード)へｉｆ変換されることを必要とする。このこと

は、コードの各命令が１つの述語を持つ述語型とされ、述語を計算処理するため１組のcompare-to-predicate演算が追加されることを意味する。ｉｆ変換プロセスが述語を計算処理するため無条件および並列ＯＲアクションのみを必要とするので、以下においては、主としてこれら２つのアクションに関して記述を行う。無条件アクションによって計算処理される述語は、単一条件に基づいて実行する命令に関して有用である。並列ＯＲアクションによって計算処理される述語は、１つのブロックの実行を複数の条件によって可能とすることができる場合に有用である。図５は、通常(normal)および補完(complement)の両モードにおける無条件および並列ＯＲアクションの実行動作を示すテーブルである。各エントリは、宛先述語に関する結果を記述する。注意すべき点ではあるが、宛先には値を割り当てられるか、不変("-"として示されている)のままとする場合もある。図５からわかるように、無条件比較演算は、常に、その宛先レジスタに１つの値を書き込む。このように、入力述語が比較演算に関するガード述語として出現するが、それは、比較演算への入力オペランドとして認識されなければならない。

【００１７】図４に示される本発明の１つの実施形態において、コンピュータ・システム１００は、パーソナル・コンピュータ、ノート・パソコン、パームトップ・コンピュータ、ワークステーション、メインフレーム・コンピュータまたはスーパー・コンピュータである。代替実施形態において、コンピュータ・システム１００は、その他のタイプのコンピュータ・システムであることもできる。例えば、コンピュータ・システム１００は、ネットワーク・サーバまたはビデオ会議システムでもあり得る。

【００１８】プロセッサ１０２はバス１０１に接続する。メモリ１０４がまたコンピュータ・システム１００に備わる。メモリ１０４はバス１０１に接続し、典型的には、プロセッサ１０２によって実行されるべき情報および命令を記憶する。メモリ１０４は、種々のタイプのメモリの形態で実施することができる。例えば、メモリ１０４は、ＲＡＭや不揮発性メモリによって実施される。更に、メモリ１０４の実施は、ＲＡＭ、ＲＯＭ、電子的に消去可能な不揮発性メモリのいずれかまたはそれらの組み合わせも可能である。

【００１９】コンピュータ・システム１００は、また、プログラム、データおよびその他の情報を記憶する大容量記憶装置１０７を含む。プログラムは、プロセッサ１０２によって実行され、プロセッサ１０２によって実行される前にメモリ１０４にダウンロードされる必要がある。

【００２０】ディスプレイ１２１は、コンピュータ・システム１００のユーザに情報を表示するためバス１０１に接続される。キーボードまたはキーパッド入力装置１２２がバス１０１に接続される。コンピュータ・システム１００の付加的入力装置は、マウス、トラックボール、トラックパッドまたはカーソル方向キーのようなカーソル制御装置１２３である。コンピュータ・システム１００に含められる可能性のあるもう１つの装置は、ハード・コピー装置１２４である。ハード・コピー装置１２４は、テキストや画像情報を紙、フィルムまたは同様のタイプの媒体に印刷するためコンピュータ・システム１００で使用される。

【００２１】コンピュータ・システム１００は、また、他の周辺装置１２６を含む。そのような他の装置１２６は、フロッピー・ディスク・ドライブ、デジタル信号プロセッサ、スキャナ、ＬＡＮまたはＷＡＮ(ワイド・エリア・ネットワーク)コントローラ、モデム、ＣＤ－ＲＯＭドライブなどを含む。コンピュータ・システム１００は、上述されたコンポーネントのいくつかを含むことなく動作する場合もある。図４は、コンピュータ・システム１００の基本コンポーネントのいくつかを示してはいるが、コンピュータ・システム１００におけるその他のコンポーネントまたはコンポーネントの組合せを排除することを意図してはいない。

【００２２】コンピュータ・システム１００は、また、(図６に示される)プログラミング言語コンパイラ２００を含む。コンパイラ２００は、プロセッサ１０２上で実行され、原始コード２０２を述語型機械コード２０２にコンパイルする。コンパイルされた述語型機械コード２０２は、次に、大容量記憶装置１０７に記憶される。述語型機械コード２０２がプロセッサ１０２によって実行されるべき時、述語型機械コード２０２は、プロセッサ１０２によってメモリ１０４に持ち込まれる。

【００２３】コンパイラ２００は、どのような種類のコンパイル・プログラムでもよい。例えば、原始コード２０１がフォートラン・プログラミング言語で書かれていれば、コンパイラ２００はフォートラン・コンパイラである。原始コード２０１がＣプログラミング言語で書かれていれば、コンパイラ２００はＣプログラミング言語コンパイラである。

【００２４】コンパイラ２００は、先ず、原始コード２０１(例えば図１Ａの原始コード)を述語型コード（例えば図１Ｃの述語型コード)に変換する。術語型実行をサポートするプロセッサ１０２に対する高品質術語型機械コードを生成するため、コンパイラ２００は、述語型コードのスケジューリングおよび割当てに先立ち、述語型コードの述語間の関係に関する情報をそのデータ・フロー分析に取り入れなければならない。

【００２５】本発明の１つの実施形態に従って、コンパイラ２００は、コンパイルされつつある述語型コードに対して述語反応データ・フロー分析を実行する。コンパイラ２００によって実行される述語反応データ・フロー分析は、(1)述語型コードのデータ・フロー特性を分析

し、(2)述語型コードの述語表現を取り扱いそして照会を行うことによって分析されたデータ・フロー特性を術語型コードに注釈として付加する。述語反応分析は述語型コードに対して直接実行されるので、オリジナルの制御フローへの対応関係は必要とされない(すなわち分析はオリジナルの原始コードに依存しない)。分析は、本発明の１つの実施形態に従うコンパイラ２００の分析機構によって実行される。

【００２６】データ・フロー特性は、演算の間の値のフローに付随する特性である。演算は、制御フローおよび術語型実行に従って条件付きで実行される。１つの例は、フロー依存分析であって、これは、１つの演算によって作成される値が後続の演算によって使用されるか否かを判断する。代替的形態としては、述語型コードは、述語型コード・ブロックの間の分岐命令をなおも含むこともできる。この場合、術語反応データ・フロー分析は、分岐による条件つき実行の存在に適応することができる。

【００２７】より具体的に述べれば、コンパイラ２００は、述語型コードにおけるcompare-to-predicate演算または命令を分析して、述語の間の関係を決定する。上述のように、これらの演算または命令は述語を計算処理するために使用される。関係は、２つの述語の互いに素のような概念、または、または述語pが別の述語qの部分集合である(すなわちpが真であれば必ずqが真である)という関係を含む。次に、これらの関係は、ユニークなデータ構造(すなわちパーティション・グラフ)として把握される。次に、このユニークなデータ構造を使用して、各データ・フロー特性が保持する実行を表現する述語表現(すなわち記号的プール表現)が表現され取り扱われる。コンパイラ２００の構造は、図７乃至図２７を参照して、以下詳細に記述する。

【００２８】図７は、図６のコンパイラ２００の詳細を示すブロック図である。図７に示されるように、コンパイラ２００は、コンパイラ２００のフロントエンド２０９に接続したｉｆ変換システム２１０を含む。コンパイラ２００は、また、ｉｆ変換システム２１０に接続した述語型コード分析システム２２０、および、述語型コード分析システム２２０に接続したスケジューラおよびレジスタ割り当て機構２３０を含む。フロントエンド２０９は、既知の機能を実行するもので、いかなる既知のコンパイラ・フロントエンドによってでも実施されることができる。ｉｆ変換システム２１０もまた、図４のプロセッサ１０２の実行形式をサポートするコンパイラのいかなる既知のｉｆ変換システムによってでも実施することができる。スケジューラおよびレジスタ割り当て機構２３０は、ＩＬＰスケジューリングを実行することができるコンパイラのいかなる既知のスケジューラおよびレジスタ割り当て機構によってでも実施することができる。

【００２９】フロントエンド２０９は、原始コード２０１(図５)を受け取り、その原始コード２０１を中間コードに変換する。中間コードはｉｆ変換システム２１０に渡される。ｉｆ変換システム２１０は、既知の方法で中間コードを述語型中間コード(すなわち述語型コード)に変換する。次に、中間述語型コードは、述語型コード分析システム２２０によって処理される。述語型コード分析システム２２０は、述語型中間コードを処理してそのコードから述語関係を抽出する。これは、原始コードを述語型コードに変換する際ｉｆ変換システム２１０が述語関係を採用するからである。述語型コード分析システム２２０は、次に、抽出した述語関係を使用して、コードのデータ依存性(すなわちスケジューリング制約または先行制約)を分析し、分析したデータ依存性をコードに注釈として付加する。言い換えると、述語型コード分析システム２２０は、ＩＬＰスケジューリングにとって臨界的な先行エッジ(すなわちスケジューリング)を正確に生成するため述語の間の関係を開拓する。そこで、コードは、注釈をつけられた先行エッジを含む注釈付き述語型中間コードになる。注釈付き述語型中間コードは、次に、ＩＬＰスケジューリングおよびレジスター割当てを行うためスケジューラおよびレジスタ割り当て機構２３０へ送られる。スケジューラおよびレジスタ割当て機構２３０は、既知の方法でＩＬＰスケジューリングおよび割当て機能を実行する。

【００３０】代替的方法として、述語型コード分析システム２２０は、逆アセンブラ(図示されていない)を経由して述語型機械コードに直接適用されることができる。そのようなシステムを使用して、例えば、異なるプロセッサを持つ異なるコンピュータ・システムに関して述語型機械コードを再スケジュールすることができる。このプロセスは目的コード変換として知られている。

【００３１】本発明の１つの実施形態に従って、述語型コード分析システム２２０は、ｉｆ変換システム２１０に直接接続して述語型中間コードを受け取る。述語型コード分析システム２２０が述語型中間コードに対して直接分析を行うので、オリジナルの制御フローに対する対応関係は必要とされない。このようにして、(1)述語型コードのスケジュージュリング制約(すなわちデータ依存性)が、スケジューリングおよびレジスタ割り当てに先行してコードに注釈として付加され、(2)分析がオリジナル原始コードの制御フローに依存しないことが可能となる。言い換えると、分析はオリジナル原始コードおよび述語型コードが生成される方法を知っている必要がない。図８は、図７の述語型コード分析システム２２０をより詳細に示す。

【００３２】図８の説明の前に用語を若干定義する。

1. 述語ブロック：直線的シーケンスの述語型演算である。述語ブロックは、一般的には、単一入口単位出口の制御フロー領域によって形成される。

2. 実行集合：Pと表記される述語pに関する実行集合は、述語pが真を割り当てられるトレースの集合である。実行トレースは、述語ブロックの実行の間、述語変数に割り当てられるブール値のレコードである。本明細書において、すべての可能な実行トレースの集合を表記するため"1"を使用する。述語pの場合、Pは、いかなる述語pに関しても1の部分集合(すなわちP⊆1)である。

3. パーティション・グラフ：パーティション・グラフは、そのノードが実行集合を表し、そのラベルをつけられたエッジがノード間のパーティション関係を表す有向非巡回グラフである。具体的には、共有ラベルrがそれらエッジのすべてが同じパーティションに属することを示すとすれば、パーティションU=M|Nは、ラベルをつけられたエッジ$U\rightarrow^{r}M$および$U\rightarrow^{r}N$によって表される。パーティション・グラフがすべてのノードがそこから到達できる先行ノードを持たないユニークなノードを含むとすれば、そのパーティション・グラフは完成品である。余計なものを減らすため、エッジは(単一パーティションを表す)同一ラベルを用いて線引きされ、従って、エッジ・ラベルが省略される。

4. 述語表現：述語表現は、実行集合を表現する記号的表現である。述語表現の基本的記号は、個々の述語に関する実行集合の名前、例えばP、Q、1などである。基本表現は、演算子和(＋)、差(－)および積(・)を使用して結合される。対応する集合演算子を基本実行集合に適用することによって形成される実行集合として述語表現は解釈される。

【００３３】図８に示されるように、述語型コード分析システム２２０は、走査機構３０１、構築機構３０３、述語照会システム３０４およびデータ・フロー分析サブシステム３０２を含む。走査機構３０１は、ｉｆ変換システム２１０(図７)から述語型中間コードを受け取るように接続される。走査機構３０１は、スケジューラおよびレジスタ割り当て機構２３０に接続するデータ・フロー分析システム３０２に接続される。構築機構３０３はまた走査機構３０１に接続している。述語照会システム３０４は、構築機構３０３およびデータ・フロー分析サブシステム３０２に接続している。

【００３４】１つの実施形態において、述語型コード分析システム２２０の構成要素３０１から３０４の各々は、ソフトウェア手段によって実施される。代替的には、述語型コード分析システム２２０の構成要素３０１から３０４の各々またはいくつかは、ハードウェアまたはファームウェア手段によって実施することもできる。

【００３５】走査機構３０１は、コードにおける述語間の局所的関係を抽出するために使用される。述語型コードの述語関係は、局所的関係および大域的関係を含む。局所的関係は、１つのcompare-to-predicate演算によって読み取られるかまたは書き込まれる述語の間の関係を指す。言い換えると、局所的関係は"cmpp"演算から来る。大域的関係はその他すべての関係を指す。一般的には、局所的関係の各々は、述語の実行集合の間のパーティション関係として記号を用いて表現される。例えば、図１のＣから観察されるように、述語pおよびqの間の関係(例えばpはqと互いに素であるという関係)は、局所的関係と見なされる。同様に、述語q、rおよびsの間の関係(例えばrとｓは互いに素で、q=r+sという関係)は局所的関係と考えられる。

【００３６】局所的関係を抽出するため、走査機構３０１は、先ず、述語型コードを連続した静的な単一の割り当て形式に変換する。これによって後続の処理が簡単になる。次に、走査機構３０１はこの形式を処理して、述語の局所的関係を検出する。図９は、走査機構３０１のプロセスの詳細を示す。

【００３７】図９において、プロセスはステップ３３０から始まる。ステップ３３１において、図８の走査機構３０１は述語型中間コードを受け取る。ステップ３３２において、走査機構３０１はコードを単一の目標順次形式に変換する。この変換は、コードにおける無条件および並列ｏｒ演算をread-modify-write順次演算に置き換える。図１０は、単一目標順次形式を例示するテーブルである。図１０からわかるように、宛先述語値は、述語入力値および比較条件のブール関数である。走査機構３０１は、既知の記述を使用してこの変換機能を実行することができる。

【００３８】ステップ３３３において、走査機構３０１はコードを静的単一割り当て形式(この形式をstatic single assignmentの頭文字をとって以下ＳＳＡ形式と呼称する)に変換する。このステップにおいて、各変数が異なるフェーズで個別に確認されることができるようにするため、走査機構３０１はコードにおける変数の名前を変更する。例えば、変数tが何度もコードの中で定義されるとすれば、走査機構３０１は、各定義毎に変数tの名前を変える(例えばt1、t2、t3などに変える)。この技法は値の番号付けと呼ばれる。値の番号付けを実行するため、各変数に関連づけられるカウンタが使用される。カウンタは各変数の定義毎に増分され、変数の連続的使用において添え字として使用される。ステップ３３４において、走査機構３０１は、順次ＳＳＡ形式の述語型コードの比較条件を正規化し、入力オペランドは辞書の順に並べられる。例えば、(a≧b)の正規化比較条件は、!(a<b)である。この場合、!は否定を意味する。正規化および値番号付けによって、補集合および合同のようなその他の関係および意味合いを単一ハッシュ関数を用いて検出することが可能となる。次に、走査機構３０１は、順次ＳＳＡ形式の正規化述語型コードをデータ・フロー分析サブシステム３０２(図８)に送る。図１１は、図１のＣの述語型コードの変換され正規化された順次ＳＳＡ形式を示す。図１１からわかるように、t1およびpは同等述語である。述語型コードの初期分析の間、

そのような同等関係が検出され、単一記号述語に対応付けされる。

【００３９】順次ＳＳＡ形式におけるコードの述語型演算のシーケンスを所与として、走査機構３０１は演算を順次走査する。比較演算が処理される場合、比較オペランドの述語記号に関連するパーティション関係が提供される。これはステップ３３５において走査機構３０１によって行われる。

【００４０】ハッシュ・テーブルは、このフェーズの間に割り当てられるソースおよび述語記号における述語名の間の対応関係を記録する。１から始まる整数がノード名として使用される。初期的には、ハッシュ・テーブルは、１にマップされる真の述語を含む。比較演算は順次処理され、ハッシュ・テーブルは、比較表現から記号名への対応付けおよび記号名から述語レジスタへの対応付けを追跡するように更新される。各比較演算毎に、右辺は、比較条件を正規化し述語オペランドを記号名と置き換えることによって、短縮される。

【００４１】走査機構３０１は、lookup_AND_stringルーチン、scan_opsルーチンおよびlookup_OR_stringルーチンを実行することによってＳＳＡ形式のコードを走査する。図１２および図１３は、局所関係を抽出するため述語ブロックを走査する上記ルーチンを示す。これらルーチンをその他の形態でプログラムすることも可能である。

【００４２】メイン・ルーチンはscan_opsであって、述語ブロックにおけるcompare-to-predicate演算に対して繰り返し動作する。遭遇するcompare-to-predicate演算毎に、比較演算右辺およびガード述語が、ハッシュ・キーの役目を果たすストリングに正規化される。lookup_AND_stringおよびlookup_OR_stringルーチンは、無条件およびｏｒ型compare-to-predicate演算からのキーをそれぞれ処理する。両方のルーチンは、比較演算によって計算処理される述語変数に対応する記号名を返す。両方のケースにおいて、計算処理のための名前がすでにハッシュ・テーブルに存在しなければ、テーブルにエントリ(すなわち項目)が作成される。ハッシュ・テーブルにエントリを追加することの結果として、新しいパーティション関係が生成される場合がある。

【００４３】図１２および図１３のルーチンを使用して、図１１に示されるコードからどのように局所的関係が抽出されるか１例を以下に示す。正規化される比較演算および削除される非比較演算を持つ図１１のコードを以下に再掲する。以下の表記で"true"は真を意味する。

p = !(a < b)・true
q = (a < b)・true
t1 = !(a < b)・true
t2 = t1 + !(c=d)・q
s = !(c=d)・q
r = !(c=d)・q

演算は順次処理され、次のような効果を発生する。

・ 初期ハッシュ・テーブルは、真を１にマップして実行集合の記号名を作成する。

・ p = !(a < b)・true。これはp= !(a < b)・1に短縮される。ルックアップの間、右辺およびその補語が初期テーブルに追加され(図１４のＡ参照)、パーティション関係1=2|3が作成される。

・ q =(a < b)・true。これは、q=(a < b)・1に短縮され、更にq=3に短縮される。

アクション：記号3に関してソース名リストにqを追加する。

・ t1 = !(a<b)・true。これはt1 =2に短縮される。アクション：記号２に関してソース名リストにt1を追加する。

・ t2 = t1 + !(c=d)・q。これはt2=2+!(c=d)・3に短縮される。ルックアップの間、"!(c =d)・3"およびその補完関係式は、ハッシュ・テーブルに追加され(図１４のＢ参照)、パーティション関係3=4|5が出力される。外部表現は更にt2 =2+4に短縮され、それはハッシュ・テーブルに追加され(図１４のＢ参照)、パーティション関係6=2|4が出力される。これはS=5に短縮することができる。

アクション：記号5に関してソース名リストにsを追加する。

・ r= !(c=d)・q。これはr=4に短縮することができる。

アクション：記号4に関してソース名リストにrを追加する。

・ 最終的ハッシュ・テーブルおよび出力された最終的パーティション関係が図１５に示されている。これらの関係は、1=2|3、3=4|5および6=2|4である。

【００４４】図８に戻ると、走査機構３０１は、述語型コードから局所的関係を抽出した後、抽出した局所的関係を構築機構３０３へ送る。構築機構３０３は局所的関係を結合することによってコードの大域的関係を決定する。例えば、1= P|QおよびQ=R|Sという関係がわかれば、pはrおよびsの両方と互いに素であることを導出することができる。コードから抽出された局所的関係を用いてパーティション・グラフを構築し、次にパーティション・グラフを完成させることによって、構築機構３０３は大域的関係を決定する。図１６は、構築機構３０３がパーティション・グラフを構築し完成するプロセスを示す。

【００４５】図１６において、プロセスはステップ３７０から始まる。ステップ３７１において、構築機構３０３はパーティションのリストという形式で局所的関係を受け取る。ステップ３７２において、構築機構３０３は、受け取った局所的関係をノードおよびエッジとして単に表現することによって初期的パーティション・グラフ４００を構築する。図１８のＡは、1=P|Q、Q=R|SおよびT2=P+Rという局所的関係を持つ図１１の述語型コード

から構築機構３０３が構築した初期パーティション・グラフ４００を示す。図１８のＡから観察できるように、すべてのノードにはルートから到達することができないので、この初期パーティション・グラフ４００は完全なグラフではない。例えばノードT2はルートから到達できない。この原因は、構築されてない制御地域に対応しそのパーティションがｏｒ型比較から作成されるノードは、初期的に到達可能ではないためである。

【００４６】ステップ３７３において、構築機構３０３は初期パーティション・グラフを完成させる。初期パーティション・グラフを完成させるため、すべてのノードがルートから到達可能となるように、既存のパーティションと整合する追加パーティションを組み入れる必要がある。構築機構３０３は、下記のようなルーチンを実行することによってパーティション・グラフを完成させる。

1. Lをパーティション・グラフのルート・ノードから到達できないパーティションのリストとする。更に、リストLにおけるパーティションはそれらが走査の間に出現した順序で発生させる。

2. Lの各パーティションをp=m|nとして、各パーティション毎に、以下を実行する。

・ lcaをmおよびnの最小共通先祖とすれば、lcaはルートから到達できる。

・ lcaは今やpおよびrに細分化できる。ここで、rはlcaに対してpの相対的補完集合である。

・ rがパーティション・グラフの中にノードとして存在しなければ(すなわちrel_compが複数のエレメントを持つノード集合を返せば)、相対的補集合を表すためノードrをルートとするパーティション・サブツリーを作成する(図１７参照)。このツリーは単純連鎖とすることができる。

【００４７】図１７のルーチンを別の形態で表すことも可能である。完成されたパーティション・グラフは、局所的関係を大域的関係に組み合わせるデータ構造であって、述語関係に関する一般的照会に応答できる形式をしている。

【００４８】上記プロセスを実行することによって、構築機構３０３は、図１８のＡの初期パーティション・グラフ４００を図１８のＢの完成パーティション・グラフ４１０に完成させる。ここで、図１８のＡの初期パーティション・グラフ４００を検証することによって、構築機構３０３は、ノードT2はノード１から到達することはできないが、その後続ノードPおよびRは(Qを経由して)ノード１から到達できると判断することができる。また、構築機構３０３はそのノード1自体がPおよびRの最小共通先祖であると判断することができる。かくして、1に関するT2の相対的補集合(すなわち1-T2)は、(1-R)-Pとして計算される(先祖Mに関するノードNの相対補集合は、MからNへのいかなる経路に沿っても、ノードの兄弟の和集合を集めることによって検出される。例えば、PおよびSは、1からRへの経路上のノードの兄弟であるので、1-R=P+Sである)。従って、1-T2=(1-R)-P=(P+S)-P=(P-P)+S=Sである。従って、1=T2|Sである。このパーティション(すなわち1=T2|S)が次にグラフに追加され、パーティション・グラフ４１０(図１８Ｂ)が完成する。ステップ３７４において、完成したパーティション・グラフが述語照会システム３０４(図８)に記憶される。プロセスはステップ３７５で終了する。図１８のＢのパーティション・グラフ４１０から観察できるように、述語pおよびsは互いに素である。

【００４９】図８に戻れば、述語照会システム３０４はデータ・フロー分析サブシステム３０２にインターフェースする。述語照会システム３０４は、構築機構３０３から受け取る完成パーティション・グラフを記憶し、そのパーティション・グラフを使用して、データ・フロー分析サブシステム３０２からの照会に回答する。データ・フロー分析サブシステム３０２は、ＳＳＡ形式の述語型コードを分析してコードのデータ・フロー特性を決定する。言い換えれば、データ・フロー分析サブシステム３０２は、(1)コードのデータ依存性、(2)上下に明確になった定義および使用、および(3)レジスタ割当ての間の有効な範囲および干渉情報を計算する。コードのデータ・フローを分析する時、データ・フロー分析サブシステム３０２は、対応する照会コマンドを述語照会システム３０４に送ることによって，コードの述語関係に関する照会を生成する。次に、述語照会システム３０４は、述語関係照会の各々が真であるか偽であるかを決定するため対応する照会ルーチンを実行させる。図１９乃至図２３は、述語照会システム３０４に関する種々の照会ルーチンを示す。これらの照会ルーチンは別の形式で実施することもできる。照会ルーチンを実行した後、述語照会システム３０４は、照会の各々が真であるか偽であるか、データ・フロー分析サブシステム３０２に通知する。

【００５０】例えば、データ・フロー分析サブシステム３０２がコードの述語pおよびqに出会う時、サブシステム３０２は、述語pが述語qの部分集合であるか否かを知る必要がある。この場合、サブシステム３０２は、is_disjoint(p, q)およびis_subset(p, q)照会コマンドを述語照会システム３０４に対して出す。次に、述語照会システム３０４は照会の各々が真であるか偽であるかを決定するため対応する照会ルーチンを実行させる。

【００５１】データ・フロー分析サブシステム３０2から出される照会コマンドは以下のものを含む。

・ true_expr()：実行の全体集合を表す述語表現。

・ false_expr()：実行の空集合を表すヌル述語表現。

これらの２つのコマンドは、(後述の)データ・フロー・ベクトルの述語表現を適切なデフォルト値に初期化するために使用される。例えば、生存性分析において、あら

ゆるベクトルは、存在しないことを表す偽の表現に初期化される。

・ lub_sum(p, ε)：P∪ε⊆ε'であるような最小ε'。

・ glb_sum(p, ε)：P∪ε⊇ε'であるような最大ε'。

これらの２つのコマンドは、ガード述語の下のポイントにおける特性を生成するために使用される。最小上限と最大下限の間の選択は、解決されるデータ・フロー問題によって定まる。例えば、生存性分析において、xがガード述語pの下で使用される演算の直後、表現εによって表される実行に変数xが存在すれば、演算の直前に、表現ε+pによって表される実行の中にxは存在する。生存性は「いかなる経路の」問題であるので、実行集合の過剰近似は保守的であり、従って、本発明はlub_sum(p, ε)を使用してε+pを計算する。「すべての経路」問題に関しては、実行集合は、過小近似化されなければならない。このように、述語表現は、glb_sum(p, ε)を使用して計算されることができる。

・ lub_diff(p, ε)：ε-p⊆ε'であるような最小ε'。

・ glb_diff(p, ε)：ε-p⊇ε'であるような最大ε'。

これらのコマンドは、ガード述語の下のポイントにおける特性を無効にするために使用される。例えば、生存性分析において、xがガード述語pの下で使用される演算の直後、表現εによって表される実行に変数xが存在すれば、演算の直前に、表現ε-pによって表される実行の中にxは存在する。生存性は「いかなる経路の」問題であるので、実行集合の過剰近似は保守的であり、従って、本発明はlub_diff(p, ε)を使用してε-pを計算する。

・ is_disjoint(p, ε)：P∩ε=0であれば真である。このコマンドは、ガード述語の下のポイントにおいて特性がなんらかの実行に含まれているかを調べるために使用される。例えば、２つの変数について、一方が他方の定義ポイントに存在すれば、それらは干渉している(すなわち異なる物理レジスタに割り当てられなければならない)。変数xが述語pの下で定義されるポイントにおいてεで表される実行に変数yが存在すれば、εがPと互いに素でなければ、それら変数は干渉している。これはis_disjoint(p, ε)を使用して検査される。

・ is_subset(p, ε)：Pがεによって表される実行集合の部分集合であれば真である。このコマンドは、ガード述語の下のポイントにおけるすべての実行に特性が含まれているかを調べるために使用される。

【００５２】述語表現の変数は、個々の述語に関する実行集合を示す記号であり、それらはパーティション・グラフのノードによって明示的に表される。述語関係に関する照会に効率的にかつ正確に応答し述語表現を取り扱うため、照会ルーチンは、述語表現を１の論理和形式(1-dnf)として知られている個々の記号の論理和に制限する。そのような表現は、記号のリストとして、または記号当たり１ビットを持つ(すなわちパーティション・グラフの中のノード当たり１ビット)を持つビットベクトルとして記憶されることができる。

【００５３】図１９のAは、記号ペアの間の論理和を検査する照会ルーチンを示し、図１９のBは、記号の間の論理和および述語表現(短縮された1-dnf形式)を検査する照会ルーチンを示す。偽表現εfalseは、単に述語記号の空らのリストである点注意する必要がある。真表現εtrueは、1を含むリストであり、パーティション・グラフのルートである。ルーチンは、PおよびQが論理和部分に含まれているようなパーティションを検出することによって、２つの記号PおよびQの間の論理和を示すことを試みる。そのようなパーティションが存在しなければ、PおよびQは相互に交わると仮定される。

【００５４】図２０は部分集合関係を検査する照会ルーチンを示す。パーティション・グラフにおいてノードPからノードQへの逆経路が存在すれば、実行集合PはQの部分集合である。部分集合関係は、支配および事後支配に対して同様であるが、述語計算の一時的順序には反応しない。

【００５５】図２１は、単一記号を述語表現に総和するルーチンを示す。表現に加えられるべき新しい記号が、表現に既に含まれている記号の部分集合である時、sum_reduceルーチンが実行される。この場合、新しい記号が追加され、結果として生ずる表現は再帰的に短縮される。

【００５６】図２２および図２３は、記号Pを述語表現から減ずるルーチンを示す。ここでは近似が必要である。記号Pが表現において記号Qの部分集合である時、Qに対するPの相対補集合でQを置き換えることによって、Q-Pが計算される。これは図２３のAのapprox_diffルーチンによって実行される。表現においてPがQと共通部分を持ち、部分集合でも上位集合でもなければ、集合の差を表すため近似計算が必要とされ、これは、図２２のBのrel_cmplルーチンによって行われる。集合差のいかなる上位集合も有効な近似である。このケースでは、PおよびQの最小共通先祖に関するPの相対補集合は、図２３のBのfind_lcaルーチンによって近似される。

【００５７】図８に戻って、データ・フロー分析サブシステム３０２が、述語反応ビットベクトル分析を実行して、制御フロー・グラフのあらゆるポイントにおいて実行集合に関する特性があるか否かを計算する。各変数は、すべてのビットベクトル内の特定の位置に対応する。各ベクトル位置は、単一のビットではなく、述語表現を持つように拡張される。演算が述語pの下の特性を生成する時、そのポイントにおけるPのすべての実行について特性が存在し、従って、集合Pがそのポイントにおいて実行集合に加えられる。演算が述語qの下の特性を無効にすれば、Qはそのポイントにおいて実行集合から差し引かれる。このケースでは、実行の何らかの部分集合に特性が存在するか否かを検査するように、制御フロー・グラフの１つのポイントにおける特性を検査する

概念が拡張される。これらの表現の取り扱いは、述語照会システム３０４に対する上述の照会コマンドを使用して実行される。

【００５８】述語反応ビットベクトル分析は、フロー分析および生存性分析を含む。フロー分析はスケジューリングのために使用され、生存性分析はレジスタ割当てのために使用される。図２４は、２つの命令の間の種々のタイプのフロー依存性を示す。既知の通り、１つのポイントからｘの使用点までxの定義を含まない経路が存在すれば、変数xは制御スロー・グラフのそのポイントにおいて生存している。フロー分析と生存性分析の相違は、フロー分析が制御フロー・グラフを前方に進むのに対して、生存性分析は制御フロー・グラフを後方に伝播する。

【００５９】述語反応生存性分析において、コードにおける変数各々毎に１つの位置を持つ述語表現のベクトルが使用される。各述語表現は対応する変数がそのポイントで生存している実行集合を表現する。各演算が逆順に取り扱われるにつれ、述語表現は更新される。

【００６０】演算において、E^-は演算の直前の表現を示すために使用され、E^+は演算の直後の表現を示すために使用される。例えば述語pによってガードされる変数xの使用を考察する。xの使用は、例えば(lub_sumを使用する)演算$E^-_x=E^+_x+P$において、演算の直前に集合Pにおけるすべての実行においてxの生存性を生成する。述語qによってガードされるxの定義は、例えば(lub_diffを使用する)演算$E^-_x=E^+_x-Q$において、集合Qにおけるすべての実行においてxの生存性を抹消する。述語rによってガードされる変数yの定義において、xが述語rの下で生存していれば、すなわち$E^+_x \cap R \neq 0$ならば、yはxを干渉する。干渉は、is_disjointルーチンを使用して検査される。!disjはis_disjointルーチンの補完を表す。

【００６１】図２５は、図１１のコードの一部に関して図８のデータ・フロー分析サブシステム３０２によって実行される述語反応ビットベクトル分析のプロセスを示す１例である。図２５は、簡略化と明示のため、図１１のコードの変数xに関する分析プロセスだけを示している。

【００６２】各演算に対応するGEN、KILLおよびTEST関数がある。これらの関数は、プログラムにおけるそのポイントにおけるデータ・フロー情報を扱う。これらの関数は、データ・フロー値に対する演算の影響を表す。各演算は、また、プログラムのそのポイントにおけるデータ・フロー値のサンプリングを行う関連TEST関数を持つ。依存分析において、TEST関数が真を返す時データ依存エッジが作成される。

【００６３】図２５からわかるように、データ・ベクトルおよび使用ベクトルが変数xに対して用意される。分析が前向きの順に実行されるので、２つのベクトルは初期的に偽に設定される。代替的に、ベクトルを初期的に真に設定することも可能である。次に、通常の定義および使用に関しGEN、KILLおよびTEST処理関数(S_4、S_5、S_9およびS_{11}という)演算に適用される。各演算において、定義は使用の前に処理される。

【００６４】演算S_4において、GEN、KILLおよびTESTが適用される。ここでは、すべての定義ベクトル位置についてデータベクトルが偽(すなわちD[S_4, S_5, S_9]=false)であるので、TESTはスキップされる。KILL処理は、データ・ベクトルの中のすべての定義ベクトル位置が偽(false)のままであると判断する。この時点で、S_4ベクトル(すなわちD[S_4])に関するデータ・ベクトル位置がPに等しいとGEN処理は判断する。次に、新しい結果がS_4に関するデータ・ベクトル位置に書き込まれる。

【００６５】S_5演算においては、GEN、KILLおよびTEST規則はデータ・ベクトルのそれぞれのベクトル位置だけに適用される。KILL処理はS_4に関するデータ・ベクトル位置(すなわちD[S_4])がPであり、S_5およびS_9の各々に関するデータ・ベクトル位置が偽のままであると判断する。GEN処理は、S_5に関するデータ・ベクトル位置がQになると判断する。TEST処理は、S_4およびS_5演算の間に依存性がないと判断する。次に、データおよび使用ベクトルが更新される。

【００６６】演算S_9において、KILL処理はD[S_4]がPであり、[S_5]はRであり、D[S_9]は偽のままであると判断する。GEN処理は、S_9に関するデータ・ベクトル位置がSになると判断する。TEST処理は、S_4およびS_9演算の間に依存性がないと判断する。TEST処理は、また、S_5およびS_9の間に出力依存性が存在すると判断する。データおよび使用ベクトルが更新される。

【００６７】演算S_{11}は変数xに関する使用演算であるので、KILL処理は存在しない。GEN処理は、S_{11}に関する使用ベクトル位置がT2であると判断する。TEST処理は、S_4とS_{11}演算の間およびS_5とS_{11}の間にフロー依存性があると判断する。TEST処理は、また、S_9およびS_{11}演算の間に依存性がないと判断する。

【００６８】次に、データ依存性の注釈がコードに加えられ、変数xに関する(図２６の)注釈付き制御フロー・グラフが取得される。次に、注釈付きコードは(図８の)スケジューラ２３０によってスケジュールされかつレジスタ割り当てされる。図２７は図１１のコードがスケジュールされたコードを示す。

【００６９】図２６および図２７からわかるように、S_4およびS_5演算は、依存性を持たないので、同じサイクルにスケジュールされる。同様に、S_9およびS_{11}演算は、依存性を持たないので、同じサイクルにスケジュールされる。結果として、同じコードに関して、従来技術では(図３のように)５サイクルを要するのに対して、本発明ではわずか３サイクルでコードは実行される。

【００７０】以上、本発明を特定の実施形態を参照して記述したが、本発明の理念および範囲を逸脱することな

く種々の修正および変更を上記実施形態に加えることが可能である点は当業者に明白であろう。従って、本明細書の仕様および図面は、例示の目的のもので、本発明をそれらに限定するためのものであると見なされるべきではない。

【００７１】本発明には、例として次のような実施様態が含まれる。

（１）述語型コードをコンパイルするコンパイラであって、述語型コードの述語表現を取り扱い照会を行うことにより述語型コードのデータ・フロー特性を分析するデータ・フロー分析システム、を備えるコンパイラ。

（２）述語型コードの述語関係を記憶する述語照会システムを更に備え、それによって上記データ・フロー分析システムがこの述語照会システムに照会して述語型コードの述語表現を取り扱うことが可能となる、上記（１）に記載のコンパイラ。

（３）上記データ・フロー分析システムが、述語表現を正確でかつ効率的に取り扱いながら、述語型コードの述語表現の結果を近似化させる、上記（１）に記載のコンパイラ。

（４）上記データ・フロー分析システムが、該コンパイラの述語変換システムと該コンパイラのスケジューラ／レジスタ割当機構の間に接続され、上記データ・フロー分析システムが分析されたデータ・フロー特性を述語型コードに注釈として付加する、上記（１）に記載のコンパイラ。

（５）上記スケジューラ／レジスタ割当機構が上記データ・フロー分析システムに直接接続され、原始コードを述語型コードに変換する述語変換システムから、述語型コードではなく上記注釈付き述語型コードを受け取る、上記（４）に記載のコンパイラ。

【００７２】（６）述語型コードをコンパイルするコンパイラにおいて使用される述語反応分析機構であって、局所的および大域的述語関係を記憶して局所的および大域的述語関係に関する照会に回答する述語照会システムを備える述語反応分析機構。

（７）述語型コードの局所的述語関係を決定する走査機構と、述語型コードの大域的述語関係を決定する構築機構と、を更に備える上記（６）に記載の述語反応分析機構。

（８）上記走査機構が、述語型コードの述語を連続した静的単一割り当て形式に変換する変換機構と、述語型コードの比較条件を正規化し、正規化した述語型コードを走査して局所的述語関係を決定する正規化機構とを含む、上記（７）に記載の述語反応分析機構。

（９）上記正規化機構が、述語型コードにおける合同および補完述語を検出する、上記（８）に記載の述語反応分析機構。

（１０）上記構築機構が、述語型コードの大域的述語関係を決定するため、上記局所的述語関係からパーティション・グラフを構築し、更に該パーティション・グラフに関するユニークなルート・ノードを形成する単一の共通先祖を作成することによって該パーティション・グラフを完成させる、上記（７）に記載の述語反応分析機構。

【００７３】（１１）上記述語照会システムが、述語型コードの第１の述語が述語型コードの第２の述語と互いに素であるか否かを決定するため上記完成パーティション・グラフを検査する第１の命令セットと、述語型コードの第１の述語が述語型コードの第２の述語の部分集合あるか否かを決定するため上記完成パーティション・グラフを検査する第２の命令セットと、を含む、上記（７）に記載の述語反応分析機構。

（１２）上記述語照会システムが、述語型コードの第１の述語を述語型コードの述語表現に追加するため上記完成パーティション・グラフを検査する第３の命令セットと、述語型コードの第１の述語を述語型コードの述語表現から差し引くため上記完成パーティション・グラフを検査する第４の命令セットと、を含む、上記（１１）に記載の述語反応分析機構。

（１３）上記データ・フロー分析システムが、述語型コードを受取り述語型コードのデータ・フロー特性を分析するため上記述語照会システムに接続され、述語型コードのデータ・フロー特性を分析する際に述語型コードの述語関係に関して上記述語照会システムに照会を行う、上記（６）に記載の述語反応分析機構。

（１４）上記データ・フロー分析システムが、述語表現を正確でかつ効率的に取り扱いながら、述語型コードの述語表現の結果を近似化させる、上記（１３）に記載の述語反応分析機構。

【００７４】（１５）述語型コードをコンパイルするコンパイラ・システムにおいて、述語反応データ・フロー分析を行う方法であって、述語型コードの述語および述語表現を取り扱って該述語型コードのデータ・フロー特性を分析するステップと、述語型コードの述語および述語表現を取り扱う際に述語型コードの述語関係を記憶する述語照会システムに対して述語型コードの述語および述語表現に関する照会を送付するステップと、上記述語照会システムから述語型コードの述語および述語表現に関する照会結果を受け取るステップと、を含む述語反応データ・フロー分析方法。

（１６）近似化の結果が正確でかつ効率的であるようにしながら、上記取り扱われる述語型コードの述語表現の結果を近似化させるステップを更に含む、上記（１５）に記載の述語反応データ・フロー分析方法。

（１７）上記述語型コードに上記分析されたデータ・フロー特性を注釈として付加するステップを更に含む、上記（１５）に記載の述語反応データ・フロー分析方法。

（１８）上記述語型コードのコンパイルが最適化されるように、上記注釈をつけられた述語コードをスケジュー

ルしレジスタ割り当てするステップを更に含む、上記（１７）に記載の述語反応データ・フロー分析方法。

【００７５】（１９）述語型コードをコンパイルするコンパイラにおいて、述語型コードを分析する方法であって、述語型コードの局所的関係を決定するステップと、述語型コードの大域的関係を決定するステップと、述語型コードの局所的関係および大域的関係を述語照会システムに記憶して、述語コードのデータ・フロー分析の間にそれら関係に関する照会が行われるようにするステップと、を含む述語型コード分析方法。

（２０）上記局所的関係を決定するステップが、上記述語型コードの述語を順次静的単一割当て形式に変換するステップと、上記述語型コードの述語を正規化し正規化した述語コードを走査することにより上記局所的関係を決定するステップとを更に含む、上記（１９）に記載の述語型コード分析方法。

（２１）上記大域的関係を決定するステップが、上記局所的述語関係に基づいてパーティション・グラフを構築するステップと、該パーティション・グラフに関するユニークなルート・ノードを形成する単一の共通先祖ノードを作成することによって上記パーティション・グラフを完成させるステップと、を更に含む上記（１９）に記載の述語型コード分析方法。

（２２）上記照会を可能にするステップが、述語型コードの第１の述語が述語型コードの第２の述語と互いに素であるか否かを決定するため上記完成パーティション・グラフを検査するステップと、述語型コードの第１の述語が述語型コードの第２の述語の部分集合あるか否かを決定するため上記完成パーティション・グラフを検査するステップと、述語型コードの第１の述語を述語型コードの述語表現に追加するため上記完成パーティション・グラフを検査するステップと、述語型コードの第１の述語を述語型コードの述語表現から差し引くため上記完成パーティション・グラフを検査するステップと、を更に含む上記（１９）に記載の述語型コード分析方法。

【００７６】

【発明の効果】本発明のコンパイラによって、述語型コードにおける述語間の関係の分析を通して最適化された目的コードが生成され、従って、述語型コードの実行時性能が最適化される。

【図面の簡単な説明】

【図１】Ａは従来型のプログラム・コードを示し、Ｂはその制御フローを示し、Ｃはそのｉｆ変換された述語型コードを示すブロック図である。

【図２】従来技術の述語型コード・コンパイラを示すブロック図である。

【図３】図２のコンパイラによってコンパイルされた後の図１Ｃの述語型コードを示す図表である。

【図４】述語型実行をサポートするプロセッサを含むコンピュータ・システムを示すブロック図である。

【図５】図４のプロセッサの比較演算の実行動作を示す図表である。

【図６】本発明の１つの実施形態に従って原始コードを述語型機械コードに変換する図４のコンピュータ・システムのためのコンパイラを示すブロック図である。

【図７】述語型コード分析システムおよび原始コードを述語型コードに変換するｉｆ変換システムを含む図６の述語型コード・コンパイラの構造を示すブロック図である。

【図８】述語型コード分析システムの構造を示すブロック図である。

【図９】図８の述語型コード分析システムの走査機構の動作の流れ図である。

【図１０】述語型コードのcompare-to-predicate演算に関する単一目標順次形式を示す図表である。

【図１１】図８の走査機構によって変換された図１Ｃの述語型コードの順次静的単一割当て形式を示すプログラム図である。

【図１２】ＡおよびＢそれぞれが、述語型コードから局所的述語関係を抽出する図８の走査機構の１つのルーチンを示すプログラム図である。

【図１３】図１２と同様に、述語型コードから局所的述語関係を抽出する図８の走査機構の１つのルーチンを示すプログラム図である。

【図１４】ＡおよびＢがそれぞれ、図１１の述語型コードに関する種々の異なる段階でのハッシュ・テーブルの１つを示す図表である。

【図１５】図１４と同様に、図１１の述語型コードに関する種々の異なる段階でのハッシュ・テーブルの１つを示す図表である。

【図１６】図８の述語型コード分析システムの構築機構の動作の流れ図である。

【図１７】相対的補集合パーティション・サブグラフを示す図表である。

【図１８】Ａが図１１の述語型コードの初期パーティション・グラフを、Ｂが完了パーティション・グラフを、それぞれ示す図表である。

【図１９】ＡおよびＢはそれぞれ、述語型コードの述語関係を照会する図８の述語照会システムの照会ルーチンの１つを示すプログラム図である。

【図２０】ＡおよびＢはそれぞれ、図１９と同様に、述語型コードの述語関係を照会する図８の述語照会システムの照会ルーチンの１つを示すプログラム図である。

【図２１】ＡおよびＢはそれぞれ、述語型コードの述語表現を取り扱う図８の述語照会システムのルーチンの１つを示すプログラム図である。

【図２２】ＡおよびＢはそれぞれ、図２１と同様に、述語型コードの述語表現を取り扱う図８の述語照会システムのルーチンの１つを示すプログラム図である。

【図２３】ＡおよびＢはそれぞれ、図２１および図２２

と同様に、述語型コードの述語表現を取り扱う図８の述語照会システムのルーチンの１つを示すプログラム図である。

【図２４】２つの命令の間の種々のタイプのフロー依存性を示すブロック図である。

【図２５】図８のデータ・フロー分析サブシステムによって図１１の述語型コードの変数ｘのフロー依存性を決定するプロセスを示す流れ図である。

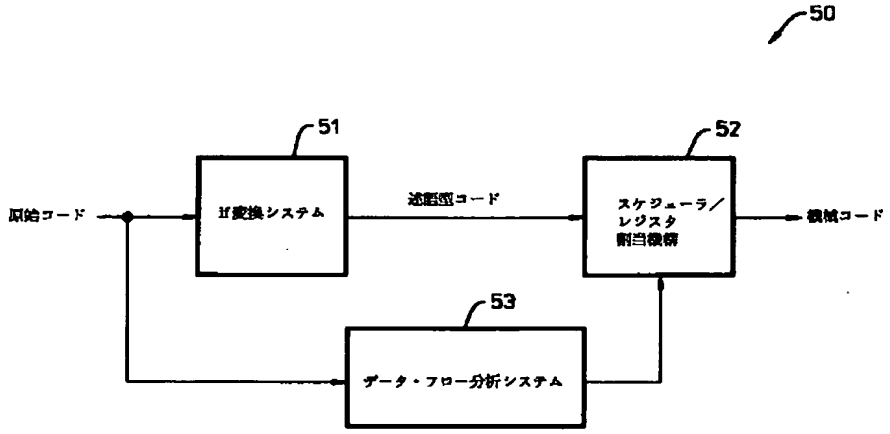
【図２６】図１１の述語型コードの変数ｘに関する制御フロー・グラフを示すブロック図である。

【図２７】本発明のコンパイラによってコンパイルされた後の図１１のコードのスケジュールされたコードを示す図表である。

【符号の説明】

５０、２００　　述語型コード・コンパイラ
５１、２１０　　ｉｆ変換システム
５２、２３０　　スケジューラ／レジスタ割当機構
５３　　データ・フロー分析システム
１００　　コンピュータ・システム
１０１　　バス
１０２　　プロセッサ
１０４　　メモリ
１０７　　大容量記憶装置
２０１　　原始コード
２０２　　述語型機械コード
２０９　　フロントエンド
２２０　　述語型コード分析システム
３０１　　走査機構
３０２　　データ・フロー分析サブシステム
３０３　　構築機構
３０４　　述語照会システム
４００　　初期的パーティション・グラフ
４１０　　完成パーティション・グラフ

【図２】



（従来技術）

【図１１】

S_1: p = ! (a<b) · true
S_2: q = (a<b) . true
S_3: t1 = ! (a<b) · true
S_4: x = . . if p
S_5: x = . . if q
S_6: r = (c!=d) · q
S_7: s = ! (c!=d) · q
S_8: t2 = t1 + (c!=d) · q
S_9: x = . . if s
S_{10}: = . .y. . if r
S_{11}: = . .x. . if t2

順次SSA形式

【図５】

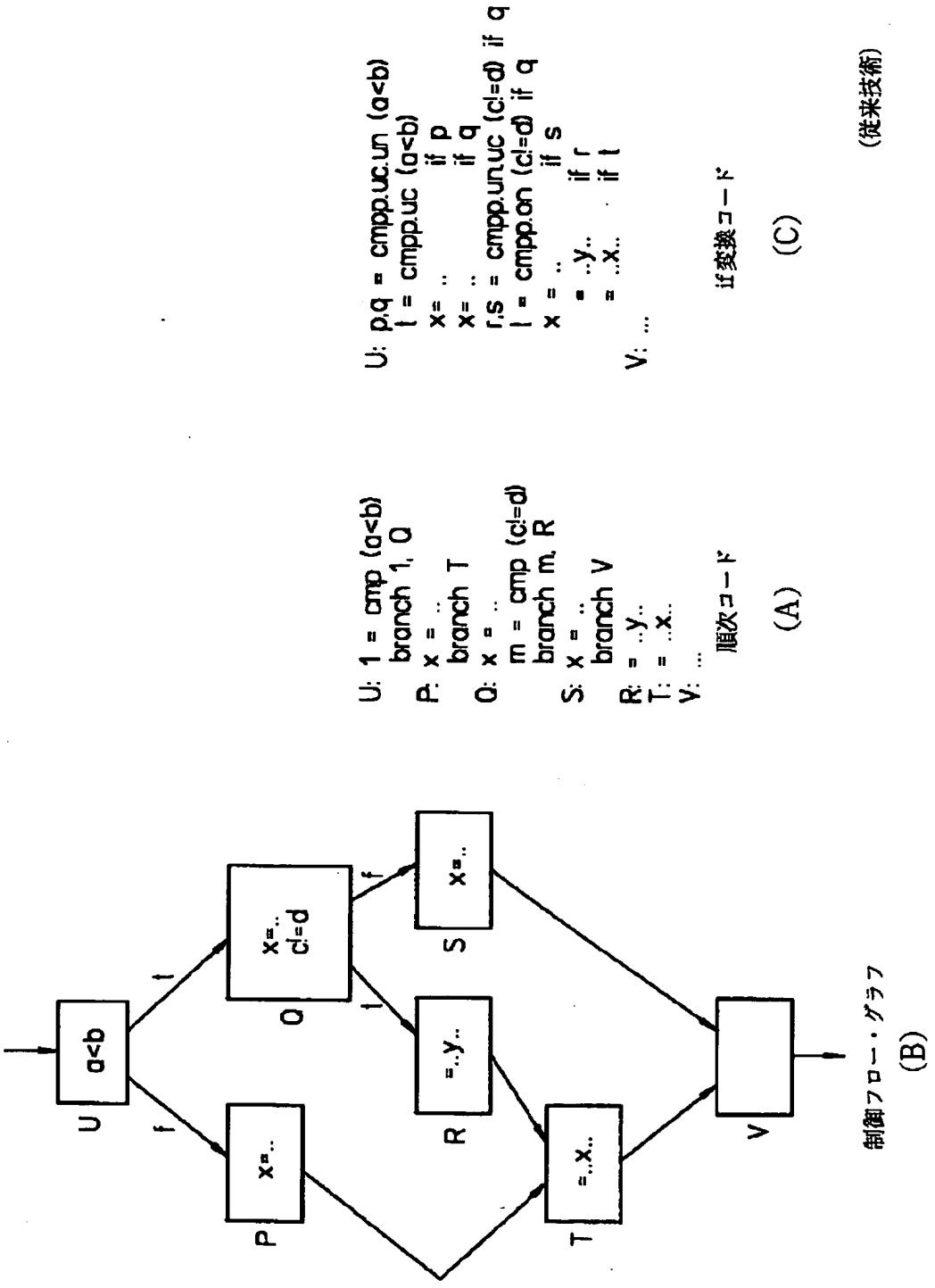
入力述語	比較結果	un	uc	cn	cc
0	0	0	0	–	–
0	1	0	0	–	–
1	0	0	1	–	1
1	1	1	0	1	–

比較演算の行動

【図１５】

記号	ストリング	ソース名
1	true	true
2	(a<b)·1	p, t1
3	!(a<b)·1	q
4	!(c=d)·3	r
5	(c=d)·3	s
6	2+4	t2

【図1】


U: 1 = cmp (a<b)
branch 1, Q
P: x = ..
branch T
Q: x = ..
m = cmp (c!=d)
branch m, R
S: x = ..
branch V
R: = ..y..
T: = ..x..
V: ...
順次コード
(A)
U: p,q = cmpp.uc.un (a<b)
t = cmpp.uc (a<b)
x= .. if p
x= .. if q
r,s = cmpp.un.uc (c!=d) if q
l = cmpp.an (c!=d) if q
x = .. if s
= ..y.. if r
= ..x.. if t
V: ...
if変換コード
(C)
U
a<b
t
f
P
x=..
Q
x=..
c!=d
t
f
R
=..y..
S
x=..
T
=..x..
V
制御フロー・グラフ
(B)
(従来技術)

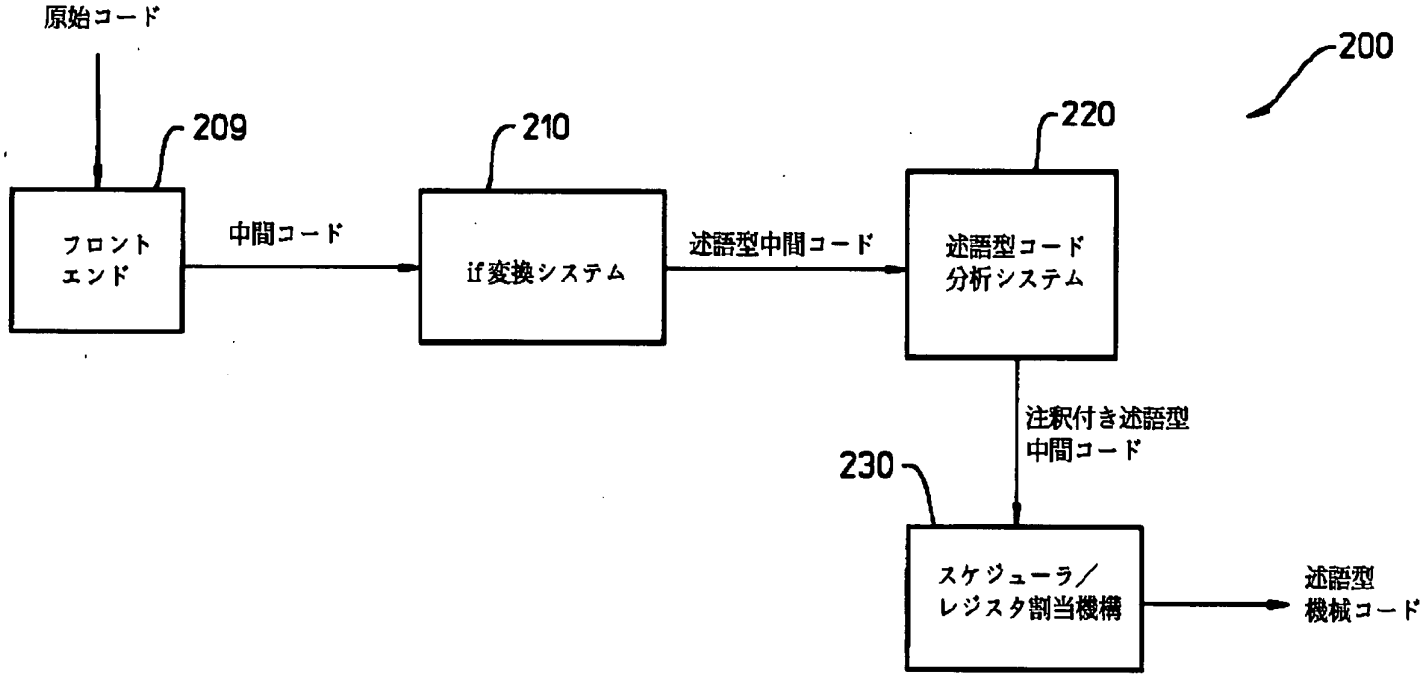
【図３】

	演算ユニット 1	演算ユニット2	演算ユニット3	演算ユニット4
サイクル 1	p,q = cmpp.uc.un (a<b)	t = cmpp.uc (a<b)		
サイクル 2	r,s =cmpp.un.uc (c!=d) if q	t = cmpp.on (c!=d) if q	x =.. if p	
サイクル 3		= ..y.. if r		x =.. if p
サイクル 4			x =.. if s	
サイクル 5				= ..x.. if t

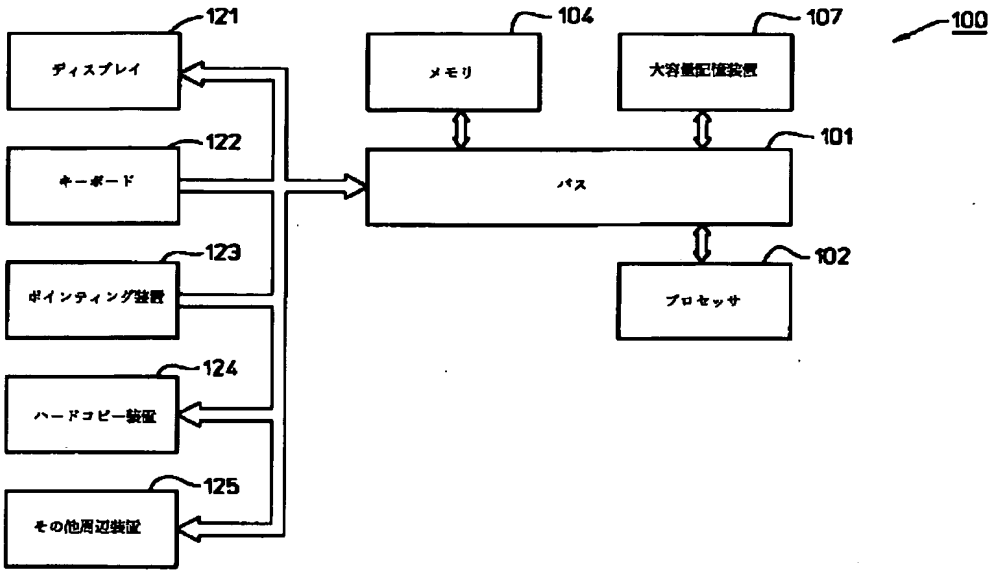
スケジュールされたコード
(コンパイル後)

(従来技術)

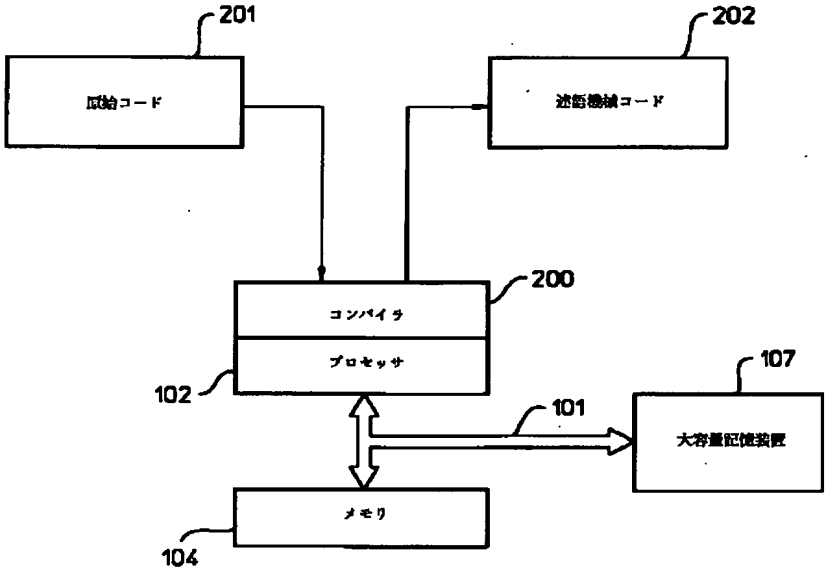
【図７】



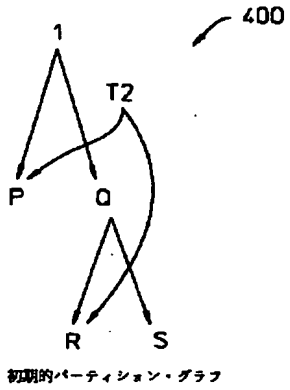
【図４】



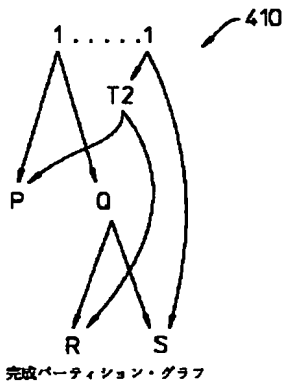
【図６】



【図１８】



(A)

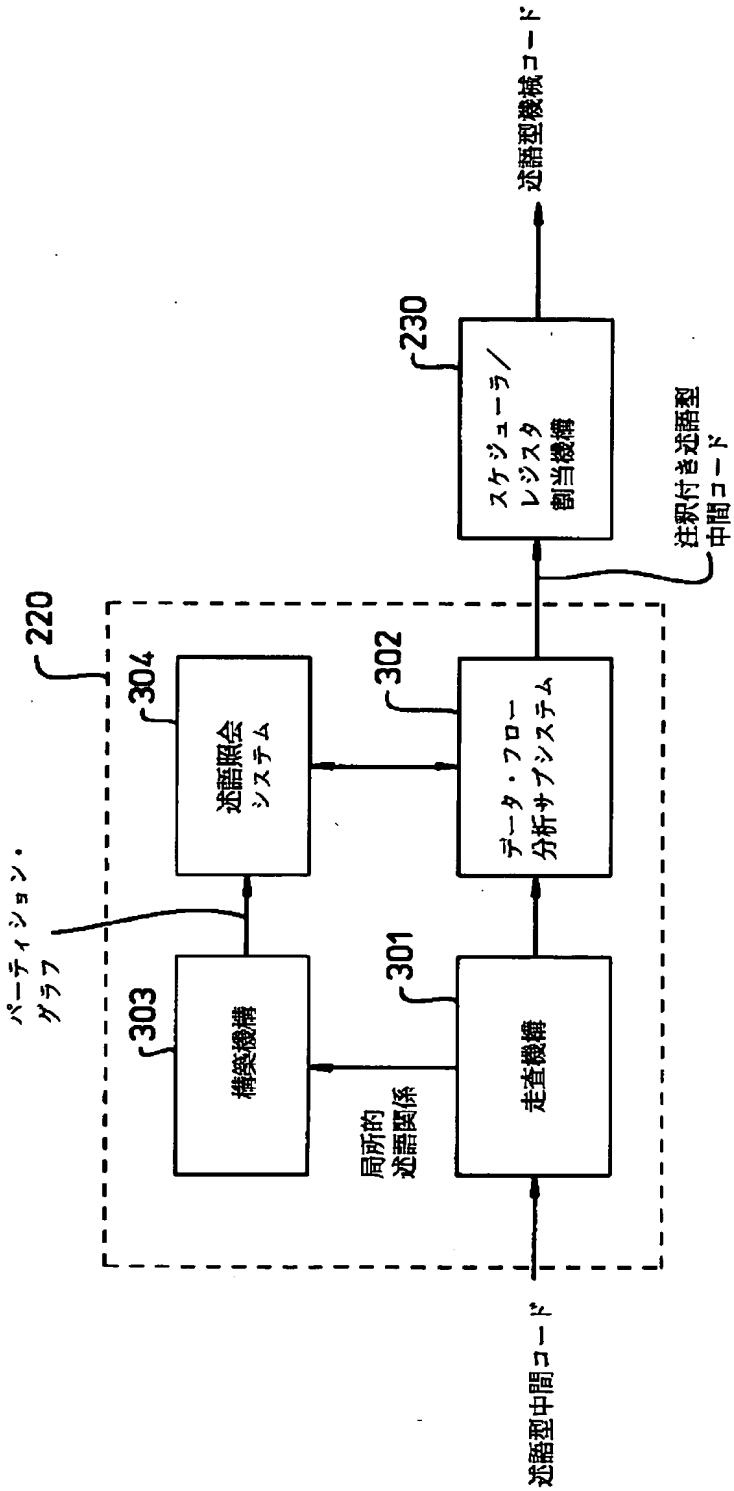


(B)

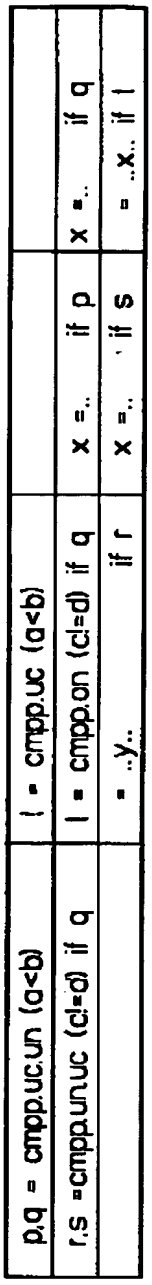
【図１０】

条件付き比較演算	順次形式
P ¬1 cmpp.un (r1 <cond> r2) if P_2	P_1= (r1 <cond> r2) · P_2
P ¬1 cmpp.uc (r1 <cond> r2) if P_2	P_1= ! (r1 <cond> r2) · P_2
P ¬1 cmpp.on (r1 <cond> r2) if P_2	P_1= P + (r1 <cond> r2) · P_2
P ¬1 cmpp.oc (r1 <cond> r2) if P_2	P_1= P + ! (r1 <cond> r2) · P_2

【図8】

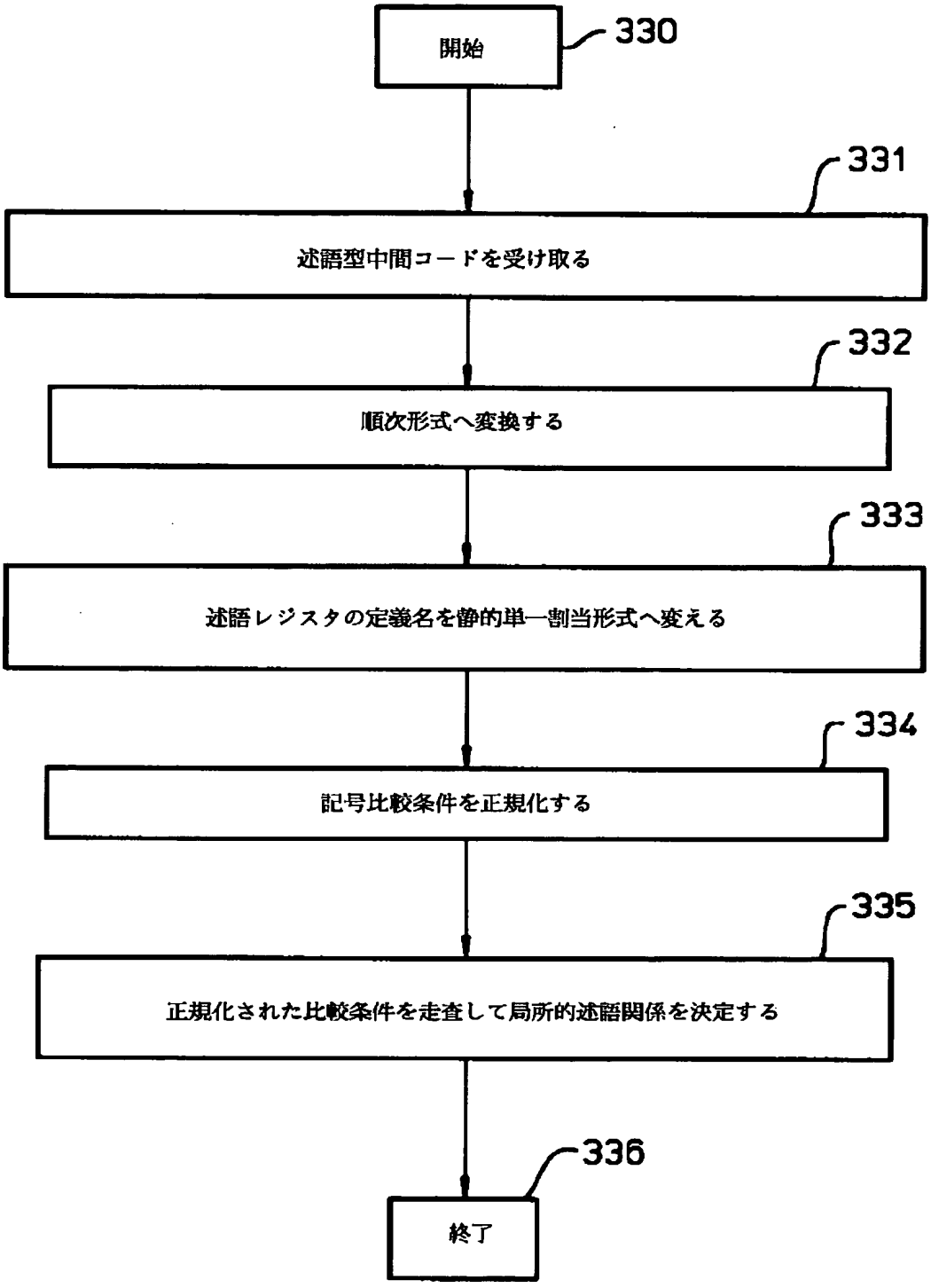


【図27】



スケジュールされたコード

【図９】


開始
330
331
述語型中間コードを受け取る
332
順次形式へ変換する
333
述語レジスタの定義名を静的単一割当形式へ変える
334
記号比較条件を正規化する
335
正規化された比較条件を走査して局所的述語関係を決定する
336
終了

【図１２】

(A)

```
lookup__AND__string (ストリング S) ルーチン
1 : //   Sは形式 [ ! ] (r1 < cond > r2) • iを持つ
2 : if  (Sがテーブルにない)  then
3 :       Sについてエントリmを作成 ;
4 :       Sの補集合エントリnを作成 ;
5 :       パーティション関係i = m | nを出力 ;
6 : endif
7 : Sに関する記号を返す
```

(B)

```
scan__ops (List  ops) ルーチン
1 : 各比較演算毎に
2 :   正規化ストリングSを短縮
3 :   if  (比較形式が無条件である)  then
4 :     m = lookup__AND__string (S) ;
5 :   else
6 :     m = lookup) __OR__string (S) ;
7 :   endif
8 :   宛先述語をmに対応付づける
9 : endfor
```

【図１３】

```
lookup__OR__string (ストリング S) ルーチン
1 : if  (Sがテーブルにない)  then
2 :   if  (Sが形式j + [i] (n1 < cond > r2) • 1を持つ)  then
3 :     S′ はSの右辺とする
4 :     k = lookup__AND__string (S′ ) ;
5 :     S = j + k
6 :  endif
7 :  //Sは形式j + kを持つ
8 : Sに関しエントリmを作成
9 : パーティション関係m = j | kを出力
10 : endif
11 : Sに関する記事を返す
```

【図１４】

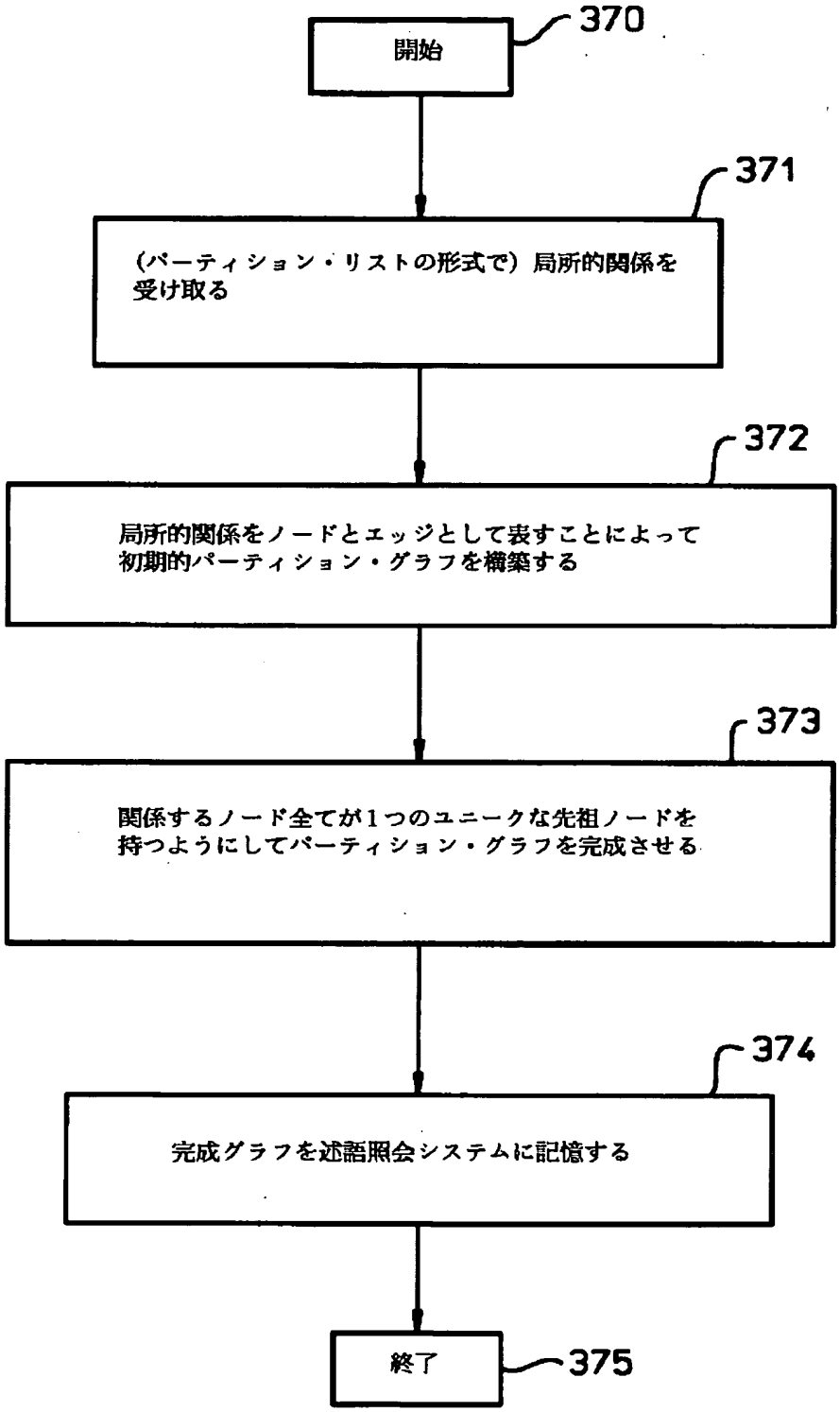
(A)

記号	ストリング	ソース名
1	true	true
2	!(a<b)•1	p
3	(a<b)•1	

(B)

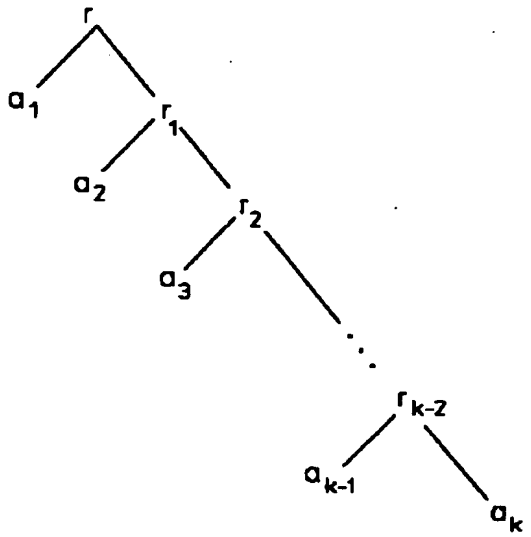
記号	ストリング	ソース名
1	true	true
2	!(a<b)•1	p, t1
3	(a<b)•1	q
4	!(c=d)•3	
5	(c=d)•3	
6	2+4	t2

【図１６】


370
開始
371
（パーティション・リストの形式で）局所的関係を受け取る
372
局所的関係をノードとエッジとして表すことによって初期的パーティション・グラフを構築する
373
関係するノード全てが１つのユニークな先祖ノードを持つようにしてパーティション・グラフを完成させる
374
完成グラフを述語照会システムに記憶する
終了
375

【図１７】

rel-comp = $(a_1, a_2, \ldots, a_k)$



【図１９】

(A)

IS_DISJOINT ルーチン１

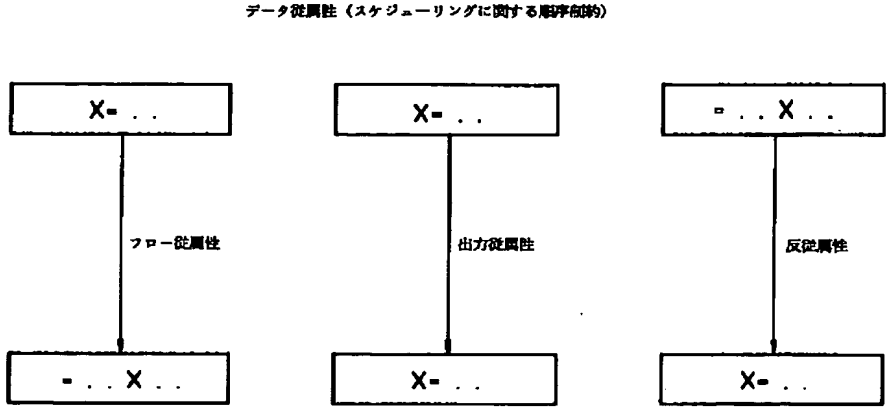
```
is_disjoint (記号 P, 記号 Q)
1 : if ∃パーティション W = X | Y | Qi
          s.t. P⊆X and Q⊆Y then
2 :       return ture ;
3 : else
4 :    return false ;
5 : endif
```

(B)

IS_DISJOINT ルーチン２

```
is_disjoint (記号 P, 表現 ε)
1 : //εは短縮されていると仮定
2 : εの各記号Q毎に
3 :    if ! is_disjoint (P,Q) then
4 :       return false ;
5 : endfor
6 : return ture ;
```

【図２４】



【図２０】

(A)

IS_SUBSET ルーチン１

```
is_subset (記号P, 記号Q)
1 : //QからPへの経路を検査
2 : Pから逆方向深さ優先探索を実行
3 : Qがあれば真を、なければ偽を返す
```

(B)

IS_SUBSET ルーチン２

```
is_subset (記号P, 表現ε)
1 : //εが短縮されていると仮定
2 : εにおける記号Q毎に
3 :     if is_subset (P,Q)  then
4 :       return false ;
5 : endfor
6 : return false ;
```

【図２１】

(A)

LUB_SUM ルーチン１

```
lub_sum (表現ε, 記号P)
1 : ε′ = ε_false ;
2 : εにおける各Q毎に
3 :     if Q⊆P then
4 :       続行 ;
5 :     else if P⊆Q then
6 :       return ε ;
7 :     else
8 :         ε′ = ε′ +Q ;
9 :     enddif
10 : endfor
11 : return sum_reduce (ε′ ,P) ;
```

(B)

LUB_SUM ルーチン２

```
sum_reduce (表現ε、 記号P)
1 : Pをεに加える
2 : 各パーティションR = P | Qi毎に
3 :     if (すべてのQiがεのメンバである) then
4 :       ε = sum = _reduce (ε,R) ;
5 :       break ;
6 :     endif
7 : endfor
8 : if (Rがεのメンバーである)  then
9 :     Rの後続ノードの全てをεから削除 ;
10 : endif
11 : return ε ;
```

【図２２】

(A)

LUB__DIFF ルーチン1

```
lub__diff (表現ε, 記号P)
1 : ε′ = ε false;
2 : εにおける各Q毎に
3 :    if Q⊆P then
4 :       続行 ;
5 :    else if P⊆Q then
6 :      ε′ = ε′ + rel__cmpl (P,Q) ;
7 :    else if (PとQが互いに素)  then
8 :      ε′ = ε′ + Q ;
9 :    else
10 :     ε′ = ε′ + approx__diff (P,Q) ;
11 :   endif
12 :  endfor
13 :  return ε ;
```

(B)

LUB__DIFF ルーチン2

```
rel__cmpl (記号P, 記号Q)
1 : //Q－Pに関する表現を返す
2 : if ! is__subset (P, Q)  then
3 :    return false__expr () ;
4 : endif
5 : QからPへの経路を検出する ;
6 : ε = ε false;
7 : 経路上の各エッジR→S毎に
8 :    R = S | Tiをエッジ R→Sを含むパーティションとする ;
9 :    各Tiをεに加える ;
10 : endfor
11 : return ε ;
```

【図２３】

(A)

LUB__DIFF ルーチン3

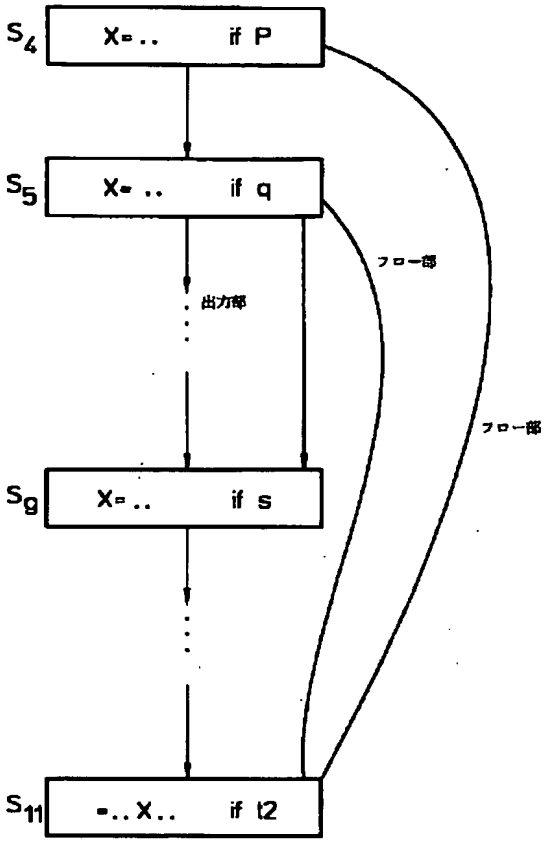
approx__diff （記号P, 記号Q）
1： //Q－Pを過大近似
2： return rel__cmpl (P, find__lca (P,Q)) ;

(B)

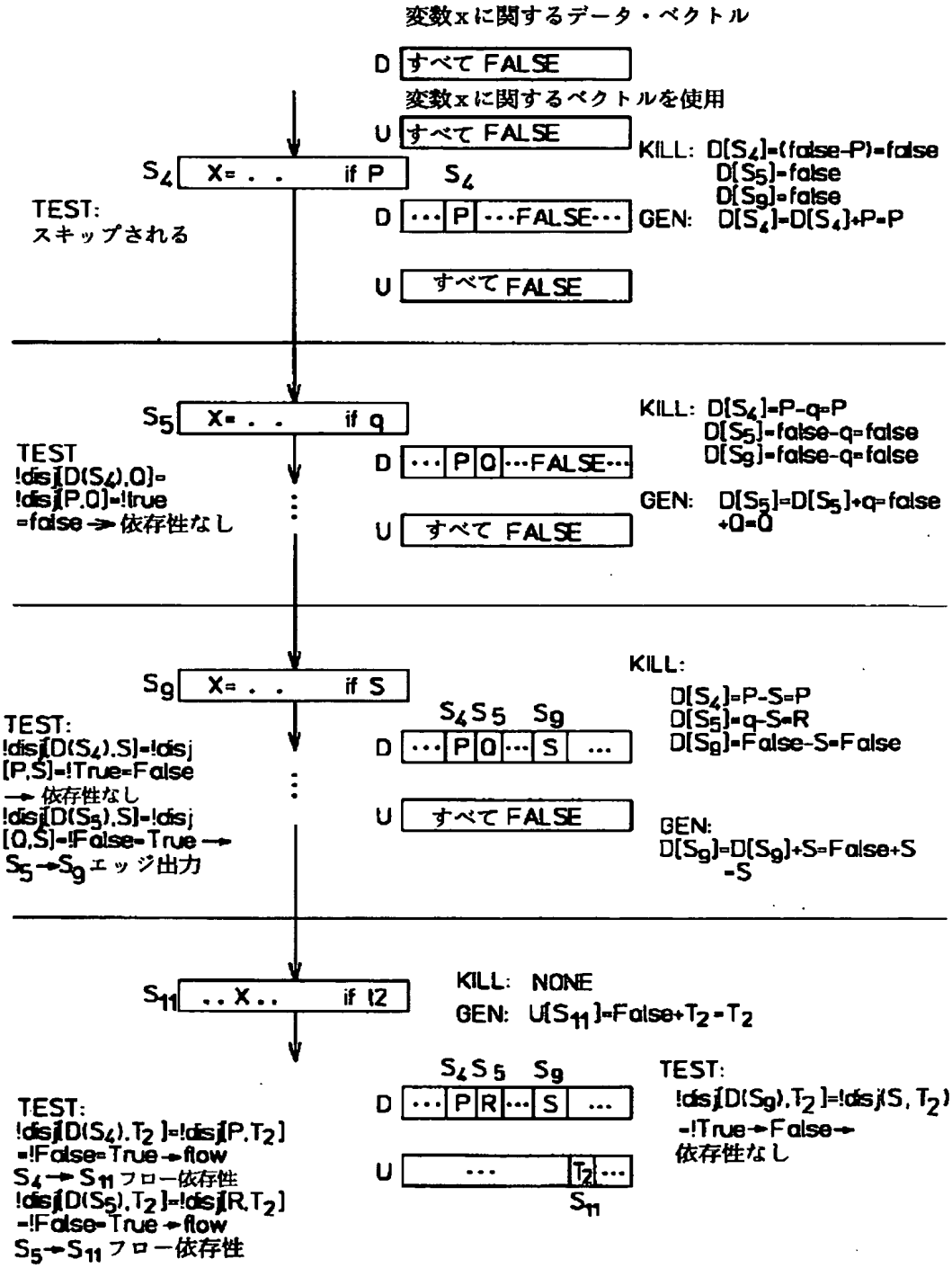
LUB__DIFF ルーチン4

find__lca（記号P, 記号Q）
1： SpをPの先祖集合とする；
2： S_QをQの先祖集合とする
3： 1から最も離れている$Sp \cap S_Q$のメンバーを返す

【図２６】



【図２５】


変数xに関するデータ・ベクトル
D すべて FALSE
変数xに関するベクトルを使用
U すべて FALSE
S4 X=. . if P
S4
TEST:
スキップされる
D ・・・ P ・・・FALSE・・・
U すべて FALSE
KILL: D[S4]=(false-P)=false
D[S5]=false
D[S9]=false
GEN: D[S4]=D[S4]+P=P
S5 X=. . if q
TEST
!disj[D(S4),Q]=
!disj[P,Q]=!true
=false → 依存性なし
D ・・・ P Q ・・・FALSE・・・
U すべて FALSE
KILL: D[S4]=P-q=P
D[S5]=false-q=false
D[S9]=false-q=false
GEN: D[S5]=D[S5]+q=false
+Q=Q
S9 X=. . if S
TEST:
!disj[D(S4),S]=!disj
[P,S]=!True=False
→ 依存性なし
!disj[D(S5),S]=!disj
[Q,S]=!False=True →
S5→S9 エッジ出力
S4 S5 S9
D ・・・ P Q ・・・ S ・・・
U すべて FALSE
KILL:
D[S4]=P-S=P
D[S5]=q-S=R
D[S9]=False-S=False
GEN:
D[S9]=D[S9]+S=False+S
-S
S11 ..X.. if t2
KILL: NONE
GEN: U[S11]=False+T2=T2
S4 S5 S9
D ・・・ P R ・・・ S ・・・
U ・・・ T2 ・・・
S11
TEST:
!disj[D(S4),T2]=!disj[P,T2]
=!False=True → flow
S4 → S11 フロー依存性
!disj[D(S5),T2]=!disj[R,T2]
=!False=True → flow
S5 → S11 フロー依存性
TEST:
!disj[D(S9),T2]=!disj(S,T2)
=!True → False →
依存性なし